UNITED STATES

PATENT APPLICATION

for

VERSIONED NODE CONFIGURATIONS FOR PARALLEL APPLICATIONS

NCR Docket No. 9900

submitted by

John Earl Merritt
Donald Raymond Pederson
and Eric Thomas Potter

on behalf of

NCR Corporation
Dayton, Ohio

Prepared by

Michael A. Hawes
Reg. 38,487

Correspond with

John D. Cowart
Reg. 38,415
Teradata Law IP, WHQ-4W
NCR Corporation
1700 S. Patterson Blvd.
Dayton, OH 45479-0001
(858) 485-4903 [Voice]
(858) 485-2581 [Fax]

Express Mail No.: EL477931803US
Date: December 7, 2001

VERSIONED NODE CONFIGURATIONS FOR PARALLEL APPLICATIONS

Background

[0001] Parallel systems employ a plurality of processors to perform tasks more quickly than would be possible with a single processor. Conventional software directing such systems breaks tasks into subtasks that can be more performed simultaneously. Parallel systems can also operate unconstrained by physical boundaries between hardware devices. For example, a parallel system can logically treat a single physical processor as two virtual processors by dividing the resources of the single processor between the two virtual entities. Virtual processors can also be allocated portions of the electronic storage capacity of the overall system in addition to a portion of the processing capacity. In such a system, if a task requires manipulation of specific data, the virtual processor that has been allocated the storage with that data is often the best choice for performing the task. Parallel system software conventionally includes substantial subportions dedicated to communications between the virtual processors.

[0002] The resources of some parallel systems are also organized on a higher level than the virtual processors. While the units of higher level organization can be given many different names, the term "nodes" will be used to discuss such units herein. Communication between virtual processors to achieve a task can entail communication between nodes when the virtual processors are associated with different nodes. Communication between the simultaneously active portions of a parallel system can become difficult when hardware or software problems cause a subset of the processing or storage resources to become unavailable. Communications can be established by repeating the procedure by which the system was initiated. During this re-initiation process, processing activity may be interrupted and progress that may have been achieved on tasks that the parallel system was addressing may be lost.

Summary

[0003] In general, in one aspect the invention includes a method for executing database transactions. A plurality of interconnected nodes are each defined in terms of processor and storage resources of a parallel computing system. A first set of virtual processors is mapped across a first subset of the nodes to create a first map with at least one virtual processor being mapped to each node in the first subset. A second set of virtual processors is mapped across a

second subset of the nodes to create a second map with at least one virtual processor being mapped to each node in the second subset. The first map is stored as a first configuration and the second map is stored as a second configuration. At least one transaction is executed using the first set of virtual processors and simultaneously at least one transaction is executed using the second set of virtual processors.

[0004] Other features and advantages will become apparent from the description and claims that follow.

Brief Description of the Drawings

[0005] Fig. 1 is a block diagram of a node of a parallel processing system.

[0006] Fig. 2 is a block diagram of a node-based parallel processing system.

[0007] Fig. 3 is a flowchart of a method for executing transactions in accordance with node configurations.

[0008] Fig. 4 is a block diagram of example node configurations.

[0009] Fig. 5 depicts example active configurations and transactions tables.

Detailed Description

[0010] The versioned node configurations technique disclosed herein has particular application, but is not limited, to large databases that might contain many millions or billions of records managed by a database system ("DBS") 100, such as a Teradata Active Data Warehousing System available from NCR Corporation. Fig. 1 shows a sample architecture for one node 105_1 of the DBS 100. The DBS node 105_1 includes one or more processing modules $110_{1...N}$, connected by a network 115, that manage the storage and retrieval of data in data-storage facilities $120_{1...N}$. Each of the processing modules $110_{1...N}$ may be one or more physical processors or each may be a virtual processor, with one or more virtual processors running on one or more physical processors.

[0011] For the case in which one or more virtual processors are running on a single physical processor, the single physical processor swaps between the set of N virtual processors.

[0012] For the case in which N virtual processors are running on an M-processor node, the node's operating system schedules the N virtual processors to run on its set of M physical processors. If there are 4 virtual processors and 4 physical processors, then typically each virtual processor would run on its own physical processor. If there are 8 virtual processors and 4 physical processors, the operating system would schedule the 8 virtual processors against the 4 physical processors, in which case swapping of the virtual processors would occur.

[0013] Each of the processing modules $110_{1...N}$ manages a portion of a database that is stored in one of the corresponding data-storage facilities $120_{1...N}$. Each of the data-storage facilities $120_{1...N}$ includes one or more disk drives. The DBS may include multiple nodes $105_{2...N}$ in addition to the illustrated node 105_1, connected by extending the network 115.

[0014] The system stores data in one or more tables in the data-storage facilities $120_{1...N}$. The rows $125_{1...Z}$ of the tables are stored across multiple data-storage facilities $120_{1...N}$ to ensure that the system workload is distributed evenly across the processing modules $110_{1...N}$. A parsing engine 130 organizes the storage of data and the distribution of table rows $125_{1...Z}$ among the processing modules $110_{1...N}$. The parsing engine 130 also coordinates the retrieval of data from the data-storage facilities $120_{1...N}$ in response to queries received from a user at a mainframe 135 or a client computer 140. The DBS 100 usually receives queries and commands to build tables in a standard format, such as SQL.

[0015] In one implementation, the rows $125_{1...Z}$ are distributed across the data-storage facilities $120_{1...N}$ by the parsing engine 130 in accordance with their primary index. The primary index defines the columns of the rows that are used for calculating a hash value. The function that produces the hash value from the values in the columns specified by the primary index is called the hash function. Some portion, possibly the entirety, of the hash value is designated a "hash bucket". The hash buckets are assigned to data-storage facilities $120_{1...N}$ and associated processing modules $110_{1...N}$ by a hash bucket map. The characteristics of the columns chosen for the primary index determine how evenly the rows are distributed.

[0016] In one implementation, nodes are defined physically, in that the processors and storage facilities associated with a node are generally physically proximate as well. For this reason, it is possible that a hardware or software problem encountered by a node will result in the unavailability of the processor and storage resources associated with that node.

[0017] Higher level groupings of resources than nodes can also be implemented. Fig. 2 is a block diagram of a node-based parallel processing system. The nodes 105_{1-12} are grouped into cliques 205_{1-4} by threes. The cliques 205_{1-4} include nodes that can be configured to share storage facilities. For example, if node 105_1 were to experience a processor failure, either of nodes 105_{2-3}, but none of nodes 105_{4-12}, can be reconfigured to access the data contained in the storage facilities that were previously associated with node 105_1. Thus, in this implementation, cliques define the atomic level of configurable storage access while nodes define the atomic level of configurable processor access. In other implementations, the atomic levels for those types of access are different. For example, the nodes within the cliques communicate with each other through the network 115. The network 115 can include software that monitors that availability of the nodes 105_{1-12} both by determining when particular nodes fail and by determining when nodes are restored or added.

[0018] Transactions performing tasks that involve manipulating certain data employ virtual processors having access to that data. Mapping functions are applied to the data and the results are used to partition the data to specific virtual processors. Using the same mapping functions, transactions can determine which virtual processors will be needed for a particular task. That group of virtual processors is identified as a transaction group. One transaction can have multiple transaction groups with a different subset of the virtual processors belonging to each transaction group. In one implementation, a transaction can establish transaction groups at different times during the execution of the transaction. For the duration of a transaction a single group identifier can be associated with the subset of virtual processors for the purposes of, e.g., insertion, processing and extraction of collectively associated data.

[0019] Fig. 3 is a flowchart of a method for executing transactions in accordance with node configurations. A mapping of virtual processors to valid nodes is generated 310. To accomplish this, nodes that are capable of reliably supporting a virtual processor, also referred to herein as

"valid nodes," are identified 312. In one implementation, an invalid or failed node is identified by the failure to send a "heartbeat" signal to the network 115. A heartbeat signal is a particular signal that is sent on a regular basis as long as operation is normal. A virtual process, or vproc, is then created 314. The vproc is associated with processor and storage resources 316. In one implementation, only processor and storage resources from a single node are associated with a single vproc. If additional vprocs are desired 318, the others are created in the same fashion.

[0020] The definitions of the vprocs are then recorded as an entry in a configuration table 320. The entry includes the association of vprocs to nodes. In one implementation, each entry or configuration is identified by an unique version number. That version number can then be used to reference a specific entry within the table of active configurations. For example, for each vproc the table entry may list the node that contains that vproc's processor and storage resources. In another implementation, the vprocs can just be listed by node. In different implementations, different amounts of information about the vprocs is stored in the configuration table entry.

[0021] After the new entry has been created the database system comes online to initiate new transaction tasks and restart existing transaction tasks 321. New transaction groups initiated for transaction tasks are assigned to the new entry 322. Thus, a transaction group initiated after the creation of a new entry will include vprocs that are defined in relation to the nodes by the new entry in the configuration table. Transaction groups that were initiated prior to the addition of the current entry of the configuration table, and have not been halted, will continue to employ vprocs in accordance with the configuration table entry that was current when that transaction group was formed 324. In one implementation, if a configuration table entry is not associated with active transaction groups, it is removed 328. As long as additional nodes do not become available 327 and the nodes do not fail 326, the current configuration table entry can be maintained.

[0022] In the event of node failure 326, the system identifies vprocs that are not affected by the failed node 329. Transaction activity corresponding to that node is halted 330. In one implementation, the transaction is reset to the last recorded state, rather than completely reset. If a transaction has initiated multiple transaction groups, the tasks assigned to a subset of the transaction group can be rolled back. For example, if one transaction group includes vprocs that

are mapped to the failed node under the configuration table entry that was current when the transaction group was initiated, the tasks for that transaction group will need to be reset. Such a transaction is also referred to herein as an "impacted transaction group." If another transaction group, however, does not include vprocs that are mapped to the failed node under the configuration table entry that was current when the transaction group was initiated, that transaction group can continue to perform tasks once the system comes back online.

[0023] The tasks being performed by impacted transaction groups are halted and the system generates another configuration using the identified, unaffected vprocs 320. The new configuration will not include vprocs mapped to the failed node until that node has been restored. Tasks that have been reset can then be assigned to transaction groups initiated in accordance with the new configuration, if those tasks do not need data that is only accessible to the vprocs that were associated with the failed node. New configurations are also generated when a node is restored or added to the parallel system 327. Those configurations are created vproc-by-vproc 310. In this way, tasks are assigned to transaction groups that have access to the processing and storage resources of all the available nodes.

[0024] In one implementation, generating a new configuration includes reassigning storage resources to nodes. For example, storage resources that were assigned to a node that experienced a processor-related failure can be assigned to another node within the same clique (as discussed with reference to Fig. 2). In one implementation, the entries in the configuration table would then be updated to include information necessary to indicate such changes in node configuration.

[0025] In one implementation, in response to availability events (for example the failure, restoration, or addition of a node), new configurations can be generated to allow the processing of new transactions or reset transactions, while non-impacted transactions continue their processing because the previous configuration was preserved. In one implementation, the detrimental effect of failures is confined to its impact on specific virtual processors and associated transaction groups.

[0026] Fig. 4 is a block diagram of example node configurations. The three configurations 410_{1-3} represent the mapping of virtual processors to nodes in three different entries of the configuration table. For example, if the parallel system is initiated with nodes A-D, the vprocs

can be mapped as shown in configuration 410_1. If node C then fails, a configuration is generated for the remaining vprocs associated with still-valid nodes. In one implementation, $vprocs_{1-4}$ are defined identically in the first two configurations 410_{1-2}.

[0027] A third configuration 410_3 is defined after node C has been restored. In one implementation, even if the third configuration 410_3 is substantially identical to the first configuration 410_1, each of the configurations 410_{1-3} will be retained in the configuration table until all transaction groups that correspond to a configuration have become inactive. It is possible, therefore, that the second configuration 410_2 will be removed from the configuration table prior to the first configuration 410_1, if all the tasks being performed by transaction groups assigned to the second configuration 410_2 are completed prior to all the tasks being performed by transaction groups assigned to the first configuration 410_1.

[0028] Fig. 5 depicts example Active Configurations and Transactions tables. The Active Configurations table includes three configurations labeled with version numbers corresponding to the order of generation. Version 101 was generated first. Initial requests for transaction groups are assigned that first version of the configuration. Version 102 is generated after the failure of node C. Transaction groups formed after that node failure are assigned to version 102. Those transaction groups will therefore not include vprocs 9-12. In one implementation, virtual processors are included as members in a transaction group based on the amount and range of data involved in a given transaction. In that case, when the node failure separating versions 101 and 102 occurs, each transaction group will be examined for viability based on whether member vprocs were associated with a failed node. In one implementation, any transaction associated with a group whose membership is impacted by this loss of virtual processors will necessarily be stopped and rolled back or reset to the nearest recorded point. Transactions that have no impacted transaction groups will continue to execute. The inuse column tracks whether active transaction groups are currently associated with the configuration table entry. When the last transaction group associated with a configuration table entry becomes inactive, the inuse column is set to no. The next administrative update of the tables will remove the transaction table entries that are not in use. In another implementation, entries are removed as soon as their are no associated, active transaction groups. In another implementation, use of entries is not tracked.

[0029] The Transaction table shows four active transactions, three of which are each associated with a single transaction group and one of which is associated with two transaction groups. Each transaction group refers to an independent association of virtual processors. The first two transactions 1024, 1025 are bound to groups 2002, 2003 that were formed during and therefore assigned to configuration version 101. When node C failed, assuming that vprocs 9-12 were assigned to that node in version 101, transaction group 2003 was impacted, due to vprocs 10 and 12. Transaction group 2002 does not include any of vprocs 9-12 and is therefore still valid. The third transaction 1026, bound to group 2004, is still valid, and does not include vprocs 9-12 because it was formed during the configuration that lacked node C. Baring subsequent failures, transaction group 2004 will run to completion. The final transaction 1027 is running tasks pursuant to a transaction group 2005 that is assigned to version 103, which reflects the recovery of node C. Transaction 1024 initiated a new transaction group 2006 assigned to version 103. In one implementation, version 102 does not include access to all storage resources. In that case transaction group 2004, does not require the unavailable resources. Transaction groups 2005 and 2006 could have been requested during configuration 102, but were deferred until version 103, because required storage resources were not available.

[0030] The text above described one or more specific embodiments of a broader invention. The invention also is carried out in a variety of alternative embodiments and thus is not limited to those described here. For example, while the invention has been described here in terms of a DBMS that uses a massively parallel processing (MPP) architecture, other types of database systems, including those that use a symmetric multiprocessing (SMP) architecture, are also useful in carrying out the invention. Many other embodiments are also within the scope of the following claims.